LING REN: So today, we're going to analyze two random algorithms we've seen in the last lecture, randomized median and randomized quick sort. But before I go into that, I'd like to make a correction in problem set two. I think you all got feedback from that.

So it's the second problem, where you're asked to combine B-trees. You have two B-trees, T1 and T2 each with some children. And we're giving you another element, k. And we ask you to combine them.

So if the two trees are the same height, how do you do that? Does anyone want to share his or her answer? Go ahead.

AUDIENCE: You put k in the middle of T1 and T2. And then you can split.

LING REN: Yeah, exactly. So you put k here. And if this is too full, you split. So why do I have to do this? Can I simply just make k my new root and do this? Go ahead.

AUDIENCE: Because T1 and T2 don't have to be valid like internal [? modular ?] nodes.

LING REN: Yeah, exactly. So for B-tree, the requirement on the root is slightly different from the requirements for the rest of the nodes. To this may have too few children. It is not a valid node. It's not a valid internal node.

But our solution actually made a mistake in the second part. In the second part, we're saying so T1 and T2, their heights are different by 1. Our solution says put k here and make a pointer. Exactly the same problem.

Does everyone see that? So this may not be a valid internal node. So what's the correct solution?

You put k here and combine it with the last child of T1. And then you may have to split and split again. OK. Everyone happy with that?

Now, today, we're going to look into randomization specifically we have seen two algorithms in class. I'll just call them quick find and quick sort. So quick find is a slightly generalized version of medium finding.

In the very first lecture and recitation, we have seen a non-randomized version of quick sort. So we divide them into groups of five. And we find a medium of each group. And then find the median of the median.

Depending on whether you are smaller or larger, we drew a funny subproblem like this. Anyone remember that? And we analyzed this runtime, where our recursion was one fifth plus 7 over 10 or something like that. And we show it's the worst case O of n.

So that's a smart algorithm, I'll give you that. But it's just complicated. You have to divide them into groups and do, well, several recursive course.

And also, let me digress a little bit, there's a very interesting point regarding this worst case O of n algorithm. Has anyone wondered why we use groups of five? Why not groups of three?

Algorithms should work in the same way, right? If we take out this first row and this last row, we can still find the median, which is just a second element in each group and find the median of the median. We still have a subproblem that looks like this. Exercise.

And it turns out if we use groups of three, when we solve the recursion, it doesn't solve to O of n. It solves to something else. OK. Now, end of digression.

Let's get back to the randomized version. So how does the randomized version work? It's that much simpler.

We have an array. Let me call it Find in array A, which is of size n. And we want to find i-th largest or smallest element in it.

So what we're going to do is that we'll pick a random element, x, in this array. And then we'll put all the smaller elements on this side and all the larger elements on that side. Now, because I'm picking the random one, so this x can be anywhere. If it is the k-th smallest element from the left, then what do I do next? My goal is to find out the i-th smallest element in this array, A.

AUDIENCE: If A is longer than i then [INAUDIBLE].

LING REN: OK. If i is less than k, then my element is on this side, right? So I should find-- OK, let me define this to be the left array, this to be the right array. I should find in the left array, what is its size? It's k minus 1. Make sense?

This is k minus 1. And that is n minus k plus one element in the middle. And so what's the last argument in that function called? It's i. So on the other hand, if i is greater than k, then I should go to my right half. Its problem size is n minus k. So what's the last argument?

AUDIENCE: i minus k.

LING REN: Agree? i minus k. OK. So, of course, if i is equal than k, then we just return x. Now, obviously, this algorithm's runtime depends on our luck, depends on this choice of k.

If k is roughly in the middle, then we reduce the problem size by roughly half. However, if k is 0 or close to n, then we only reduce the problem size by a little bit. So it's impossible to give a definite runtime.

So what we opt to do in randomized algorithm is that we analyze expected runtime. What does that mean? So we can write the recursion of this. It's T of n equals-- I have two subproblems.

One of them is T of k minus 1. The other is T of n minus k. So which one should I put into the recursion?

AUDIENCE: [INAUDIBLE].

LING REN: Yeah. I don't know, right? I don't know whether my element is on a left or on the right. So I'll be conservative and take a max of these two. Let me write it down a little bit.

And I have some amount of work to do before I go to my subproblem. What's the complexity of that work? Go ahead.

AUDIENCE: [INAUDIBLE].

LING REN: It's all O of n. O of theta n. Why is that?

AUDIENCE: Because you have to create the array.

LING REN: Yes. Because you have to scan the array once to put the smaller elements on one side and the larger elements on the other side. Now, this recurrence is impossible to solve, because I don't know what k is. So instead, we'll just calculate its expectation.

So the expectation of T of n is taking average over all the randomness, which means the choice of k. So there is a probability that my k is equal to j. If my k is equal to j, then I should take the maximum of-- sorry.

If my k is equal to j, then I should take the expectation of the maximum between those two. And according to the definition of expectation, I should do a sum from j equals 0-- no, not zero. I'm starting with 1 all the way to n, right? Any questions so far?

Now, obviously, depending on my choice of j, sometimes this one is larger. Sometimes that one is larger. I'll just write it a little verbosely.

So if my j is 1, then I should take the right one. Plus, if j is 2, then I should take m minus 2, so on and so forth, right? I think everyone gets this, right?

So I'll just directly jump to the next step. So when j is smaller than half of n, I will take the right one. If j is larger than half of n, I will take left one. And they happen to be symmetric. And what I have here is-- everyone gets that?

Now, maybe I'm missing the minus 1 here. OK, I shouldn't sum to there. Sorry. I have n minus 1, n, minus 2, n minus 3, all the way to half of n. But from there, I know longer go down. I go backwards, half of n plus 1 plus 2 plus 3, all the way back to n minus 1. Any questions so far?

Oh, we forgot our last term, which is a theta n. Now, how do we solve this? Any thoughts? This is a recurrence on the expectation of T.

So for this type of general recurrence, we don't have a very good way. Instead, what we'll do is just take a random guess, and then see if it is correct. So I don't really need to guess in this case, because I know it's O of n.

So let's just assume our expectation of Tn is theta of n. What does that mean again? It means I can find some constant, such that this expectation is bounded by a constant times n. So far so good?

Now, we can use induction, assume that this holds for everything up to n minus 1. And we're going to show this also holds for n. Then we're done, right?

Now, we'll just plug that in. The expectation of T n will be less or equal than this sum from half of n to n. Right? Yeah, I just plugged that in. Of course, plus our theta n term.

Now, what's the sum of this guy? Any guesses? n square? n cube? or n? OK. It's probably a messy.

More cleanly, I can pull this B out. It's just the sigma of j if I change my sum, decrease the range of sum by 1. What is the sigma some of j? What order first?

AUDIENCE: n square.

LING REN: n square. OK. Yeah, definitely n square. But we need to be a little bit more precise than that. So what's the coefficient before the n square? So I claim this coefficient is 3 over 8. Can anyone see that?

AUDIENCE: Why did you assume that the expected value is theta n.

LING REN: Oh, that's just a guess. If it's wrong, we'll have to assume something else, which we'll see in the next example. But good question. OK. Yeah. Let me ask the question again. I claim this sum is roughly 3 over 8 n square. Can anyone see that? Any ideas?

So I don't know how to calculate this term. But I do know how to calculate sigma from 1 to n, right? This is easy. What's that?

It's half of n, n minus 1. So it's roughly half of n squared. Now, this term is the sum of this minus the sum to half of n. So it's roughly half of n squared minus one half of one half n squared. Makes sense?

So this is roughly 3 over 8 n squared plus an order n term or less, or constant. Any questions? Does that makes sense?

Then it's very easy if we just plug this in. Sorry. There is a mistake. I just realized. Can anyone point that out?

So how many terms do I have in total? One from n, I have [? a ?] term. So each term should appear twice. Correct? So I should have a two here.

And so I somehow just throw away this probability. But this probability is 1 over n. Because I'm choosing a random element, there is 1 over n probability that is equal to 1, equal to 2, 3, 4. Every of this is 1 over n. Correct?

So I should have 2 over n here, 2 over here. And if we plug this in, it's 3 over 8 n cubed plus a theta n. Our goal is to show this is less than B times n, which is clearly true.

Because this is 3/4 of n, 3/4 of B times n, plus another term. We can say this is another constant D times n. And if we choose B accordingly, this can hold. Any questions?

You look confused or too easy. OK. Our guess is the latter. Oh, it is not? OK? So then we have solved this expected runtime of quick find.

Now, let's look at quick sort. Quick sort is very similar. The only difference is that once I put all the smaller elements on one side and the larger elements on the other side, instead of going into one of them, I have to sort of both.

So the only change is that instead of taking the max here, I need to add them. Correct? So now the same thing here. Instead of taking the max, I should add them up.

Now when it propagates here, so every term appears twice all the way from j equals 1 to j equals n. Is everyone following?

AUDIENCE: Can you repeat that?

LING REN: Hm-hmm?

AUDIENCE: Can you repeat that?

LING REN: OK. Sure. Originally, we have a max here. So first, did everyone get this part? We have a plus instead of a max here. We have to solve both the problems.

Now, if it's a max, then what I have is n minus 1, n minus 2, all the way to half of n, and then half of n, half of n plus 1, all the way back to n minus 1, right? And if I have a sum, then what I have is for j equals 1's case, I have T of 0 and T of n and minus 1. This is j equals 1. If j equals 2, I have T of 1 and T of n minus 2.

As j increases, this one goes from 0 to n. And this one goes from n minus 1 to 0. I'm going to sum all of them up. Does that answer your question? OK. So instead of from half of n to n, we're summing from 1 to n.

Now, we also have another good question here. Why do I guess it's theta n? Well, it's just a random guess. It could be wrong. For example, in this case, it's just incorrect.

Why? Because every sum, the range becomes a 1 to n. Now what I have is no longer 3 over 8 over n. What do I have again? What do I have now if it's 1 over n?

AUDIENCE: One half.

LING REN: Yeah. It's one half of n. But if I change one half here, what I get is B times n plus D times n. I want to prove that it's smaller than B times n, which is clearly impossible no matter how you choose B.

Did everyone get that? If we the same assumption, I make the hypothesis and we plug them in, we can no longer prove the induction step. OK. So what we do? We make another guess.

So let me rewrite our recursion. So what's the next guess? Any guesses? How about we just guess n square? Anyone unhappy with that guess?

So we can do the same thing. We can plug it in. And that will be a-- sorry, I missed another term. This is 1 over 2 here.

If we make that guess, then what we have is the sum from 1 to n minus 1, the sum of j square plus a theta n. And the sum of j squared is roughly n cubed divided by 3. Is that obvious to everyone?

Maybe not. OK. Can anyone explain this to us, why can I claim the sum of square term is n cubed over 3? Go ahead.

AUDIENCE: It's n times n minus 1 times n minus 2 over [INAUDIBLE].

LING REN: I think you're correct. There is a formula, which is-- yeah, I don't remember exactly, but it's roughly this. If you know this, then you definitely see that. If you don't, we can turn this sum into an integral. And that is n cubed over 3. Make sense?

If we plug that in, what do we have? 2 over n 3n cubed plus theta n. Of course there's a B here. And we want to show this is less than B times n squared.

Does it hold? Is it true? It's clearly true, right? So this-- no. This is cubed. Yeah. Sorry, I am making many mistakes. But that's actually good to catch your attention.

But it actually worries me that you didn't point this out. This is 2/3 n squared times B. It's clearly less than B n square. OK. So we claimed the algorithm is n squared. Correct? Go ahead.

AUDIENCE: Does that mean that you claim that it's less than n squared, being that it's n squared definitely?

LING REN: Exactly. So what I've proved here is that the algorithm is definitely O of n square, but maybe less. And you can see we still have a lot of room here. This inequality is not very tight.

So in fact, it's a very good question, how do we make that guess. So you already know the answer is n log n, right? So it's not interesting. But if you don't, then how do we go about and do things? We have to make these guesses.

So how about we know this n doesn't hold and 2 is too much. Next guess is n raised to 1 plus epsilon. Then what will we have?

If we carry out the same integral argument, we have 2 plus epsilon, n raised to 2 plus epsilon over 2 plus epsilon. Correct? And if we plug that in, we get this. And we want to show it's less than n raised to 1 plus epsilon.

Does this hold? This term is less than 1. And that's n raised to 1 plus epsilon. So this is true. So we can easily prove it's, indeed, n raised to 1 plus epsilon for any epsilon. Questions?

But is it tight? We still don't know. So then what do we do? We just make another guess. And let's guess T of n is [INAUDIBLE] n log n.

Definitely, you may run into two cases. You can either prove it or not. If this doesn't hold, you just go to log n square. And gradually, you will find the answer.

Well, if you don't know the answer, probably this is how you do things. Now, if we guess this n log n, then I have a little harder equation here. Because it's now j log j.

How do I compute that? Yeah. It's not the sum of natural number or squares of numbers, so you cannot use a formula like this. But we can still use the integral argument.

I'm not going to do that, because that's what you should have learned in calculus or other math class. But it happens that this integral of j log j is roughly half of n squared log n minus some constant times log n. I think you can change this constant. But it's roughly smaller than that.

If you plug that in, you get n over 2 one half times B plus theta n. And we want it to be smaller than B times n log n, which will be true. This is exactly that, but we are minus some term. And the term we are extracting is larger than the theta n term.

So we can prove this algorithm is n log n. But you can ask the same question, how do I know it's n log n? Or if I don't know it's n log n, maybe I should go about and try log log n. So you're welcome to try.

And it's actually a very good thought. Because I think is very uninteresting if you already know the answer. If you don't know, you have to try that.

But when do you stop? At a reasonable point, you can also prove the other way that it's runtime is larger than something. Here, if we prove this big O of n log n, if you can show the other way that it's omega n log n, then you know you have arrived at a final answer.

That's the math part. Any questions about that? Yeah. Any questions about everything I have said so far? If not, so lastly, I just have a few comments, several terminology.

Now, this recitation we focused on expected runtime. You have already seen amortized runtime. Or you may have heard of average runtime.

So to be honest, expected and amortized are just too fancier ways of saying average. But in algorithm analysis, we do mean slightly different things with these terms. So the difference is that we are averaging over different things.

So if we say average runtime, we usually mean taking the average over input. We can imagine a quick sort of quick find algorithm that doesn't use randomness, where you'll always select your first element as your favorite. That's a reasonable algorithm.

If your input is random, then you can carry out the same argument and show that its complexity is over n or n log n. But if your input is pre-sorted or reverse sorted in some special cases, you cannot do that. So average runtime is usually a very weak argument, because you have to make assumptions about your input.

And expected runtime is definitely better, because we are taking average over the randomness we introduced. They are independent of the input. So we're not making any assumptions on the input.

So of course, this comes at a price. This randomness doesn't come for free. So in fact, it's very hard to generate high quantity random numbers. Maybe at the end of the class, you will see that in crypto, a lot of works are just devoted to generate high quantity random numbers. If you can do that efficiently, you can actually solve a lot of problems.

So amortized runtime are slightly, again, different from these two, because they are taking average over a number of operations. You're doing many, many operations in a row. And some of them takes longer. Some of them take shorter. And you take average on those.

OK. That's this recitation. Any questions? OK.